

5 CLAIMS:

- 10 1. A method of watermarking a software object whereby a watermark is stored in the state of the software object as it is being run with a particular input sequence.
2. A method as claimed in claim 1 wherein the software object may be a program or piece of program.
- 15 3. A method as claimed in any one of claims 1 or 2 wherein the state of the software object may correspond to the current values held in the stack, heap, global variables, registers, program counter and the like.
- 20 4. A method as claimed in any preceding claim wherein the watermark is stored in an object's execution state whereby an input sequence *I* is constructed which, when fed to an application of which the object is a part, will make the object *O* enter a state which represents the watermark, the representation being validated or checked by examining the stack, heap, global variables, registers, program counter and the like, of the object *O*.
- 25 5. A method as claimed in any one of claims 1 or 2 wherein the watermark is embedded in the execution trace of the object *O* whereby, as a special input *I* is fed to *O*, the address/operator trace is monitored and, based on a property of the trace, a watermark is extracted.
- 30 6. A method as claimed in any one of claims 1 to 4 wherein the watermark is embedded in the topology of a dynamically built graph structure.
- 35 7. A method as claimed in claim 6 wherein the graph structure (or watermark graph) corresponds to a representation of the data structure of the program and may be viewed as a set of nodes together with a set of vertices.

FOUO "SECRET"

8. A method as claimed in any preceding claim further comprising building a recognizer  $R$  concurrently with the input  $I$  and watermark  $W$ .

9. A method as claimed in claim 8 wherein  $R$  is a function adapted to identify and extract the watermark graph from all other dynamic structures on the heap or stack.

10. A method as claimed in either claim 8 or 9 wherein the watermark  $W$  incorporates a marker that will allow  $R$  to recognize it easily.

11. A method as claimed in any one of claims 8 to 10 wherein  $R$  is retained separately from the program and whereby  $R$  inspects the state of the program.

12. A method as claimed in any one of claims 8 to 11 wherein  $R$  is dynamically linked with the program when it is checked for the existence of a watermark.

13. A method as claimed in any preceding claim wherein the application of which the object forms a part is obfuscated or incorporates tamper-proofing code.

14. A method as claimed in any one of claims 8 to 12 wherein  $R$  checks  $W$  for a signature property  $s(W)$ .

15. A method as claimed in claim 14 wherein the signature property  $s(W)$  is evaluated by testing for a specific result from a hard computational problem.

16. A method as claimed in either claim 14 or claim 15 including the creation of a number  $n$  which may be embedded in the topology of  $W$ , whereby the signature property may be evaluated by testing one or more numeric properties of  $n$ .

17. A method as claimed in claim 16 wherein the signature property is evaluated by testing whether  $n$  is the product of two primes.

AMENDED SHEET  
IPEA/AU

00033424

18. A method of verifying the integrity or origin of a program including:  
watermarking the program with a watermark  $W$ , wherein the watermark  $W$  is  
stored in the state of a program as the program is being run with a particular  
input sequence  $I$ ;  
building a recognizer  $R$  concurrently with the input  $I$  and watermark  $W$  wherein  
the recognizer is adapted to extract the watermark graph from other  
dynamically allocated data wherein  $R$  is kept separately from the program;  
wherein  $R$  is adapted to check for a number  $n$ .
19. A method of verifying the integrity of origin of a program as claimed in claim 18,  
wherein  $n$  is the product of two primes and wherein  $n$  is embedded in the  
topology of  $W$ .
20. A method as claimed in either claim 18 or 19 wherein the number  $n$  is derived  
from any combination of numbers depending on the context and application.
21. A method as claimed in any one of claims 18 to 20 wherein the program or  
code is further adapted to be resistant to tampering, preferably by means of  
obfuscation or by adding tamper-proofing code.
22. A method as claimed in any one of claims 18 to 21 wherein the recognizer  $R$   
checks for the effect of the watermarking code on the execution state of the  
application thereby preserving the ability to recognize the watermark in cases  
where semantics-preserving transformations have been applied to the  
application.
23. A method of watermarking software including the steps of:  
embedding a watermark in a static string; and  
applying an obfuscation technique whereby this static string is converted into  
executable code.
24. A method of watermarking software wherein the watermark  $W$  is chosen from a  
class of graphs  $G$  wherein each member of  $G$  has one or more properties,  
such as planarity, said property being capable of being tested by integrity-  
testing software.

25. A method of watermarking software as claimed in claim 24 wherein the watermark is rendered tamperproof to certain transformations by subjecting the watermark graph to one or more local transformations.

26. A method of watermarking software as claimed in claim 25 wherein each node of the watermark graph is expanded into a cycle.

27. A method of fingerprinting software wherein a plurality of watermarked programs obtained as claimed in any preceding claim are produced.

28. A method of fingerprinting software as claimed in claim 27 wherein the watermarked programs each of which has a number  $n$  with a common prime factor  $p$ .

29. A method substantially as herein described with reference to the drawings.

30. Software written to perform the method as claimed in any preceding claim.

31. A computer programmed to perform the method as claimed in any one of claims 1 to 27.